Monthly Notices
of the
ROYAL ASTRONOMICAL SOCIETY

MNRAS **478,** 3149–3158 (2018)
Advance Access publication 2018 May 18

doi:10.1093/mnras/sty1310

ELT-scale adaptive optics real-time control with the Intel Xeon Phi Many Integrated Core Architecture

David R. Jenkins,* Alastair Basden and Richard M. Myers
CfAI, Department of Physics, Durham University, DH1 3LE, UK

Accepted 2018 May 14. Received 2018 May 14; in original form 2018 February 1

ABSTRACT

We propose a solution to the increased computational demands of Extremely Large Telescope (ELT) scale adaptive optics (AO) real-time control with the Intel Xeon Phi Knights Landing (KNL) Many Integrated Core (MIC) Architecture. The computational demands of an AO real-time controller (RTC) scale with the fourth power of telescope diameter and so the next generation ELTs require orders of magnitude more processing power for the RTC pipeline than existing systems. The Xeon Phi contains a large number ($\geq$64) of low-power x86 CPU cores and high-bandwidth memory integrated into a single socketed server CPU package. The increased parallelism and memory bandwidth are crucial to providing the performance for reconstructing wavefronts with the required precision for ELT scale AO. Here, we demonstrate that the Xeon Phi KNL is capable of performing ELT scale single conjugate AO real-time control computation at over 1.0 kHz with less than 20μ s RMS jitter. We have also shown that with a wavefront sensor camera attached the KNL can process the real-time control loop at up to 966 Hz, the maximum frame-rate of the camera, with jitter remaining below 20 μ s RMS. Future studies will involve exploring the use of a cluster of Xeon Phis for the real-time control of the MCAO and MOAO regimes of AO. We find that the Xeon Phi is highly suitable for ELT AO real time control.

Key words: instrumentation: adaptive optics – methods: numerical.

1 INTRODUCTION

Ground based optical and near-infrared astronomical telescopes suffer image degradation due to the optical aberrations introduced by the Earth's turbulent atmosphere. Adaptive optics (AO, Babcock 1953) is a technique that has been successfully deployed to help reduce the effect that the atmosphere has on scientific observations. The basic operation of AO involves detecting the shape of an incoming wavefront using a wave-front sensor (WFS) and using this information to correct for the atmospheric perturbations using an optical correcting element, usually a deformable mirror (DM). The wavefront measurements are normally approximated by measuring the local wavefront slope at a number of discreet points in the pupil plane and the approximate wavefront phase can be reconstructed from these local slope measurements using a typically computationally intensive approach.

The atmosphere is constantly changing, and so adaptive optics systems need to operate at rates consistent with the rate of change of the atmospheric conditions. This is characterized by the atmospheric coherence time, τ_0, typically of order 1 ms at optical wavelengths of around 500 nm. Not only should the exposure time of the WFS be $<\tau_0$ to capture a 'snapshot' of the turbulent phase, but the correction also needs to be applied within this time window otherwise the turbulent wavefront will have evolved past the point of measurement by $\geq$1 rad of wavefront error and the correction after this time will no longer be valid.

1.1 AO systems

An adaptive optics real-time controller (RTC) is the hardware and software responsible for ensuring that the AO correction is computed and applied at the required rate for the atmospheric conditions (Fig. 1), typically of order 1 kHz for visible wavelengths, such that correction is applied within an atmospheric coherence time. The RTC takes wavefront sensor information (images) as input, calculates local wavefront slopes, computes the residual wavefront shape, and converts this to the required commands to be sent to the DM. The required high frame rate places strict requirements on the performance of the RTC so that it can meet the demands set out by the requirements of the AO instrument. Additionally, the variation in frame rate (the jitter) must also be low, otherwise scientific performance begins to suffer. An AO system's requirements are mostly defined by the parameters that quantify the strength and conditions of the atmospheric turbulence present at the site of observation, by the telescope size, and by the AO system type. These parameters can

* E-mail: d.r.jenkins@durham.ac.uk

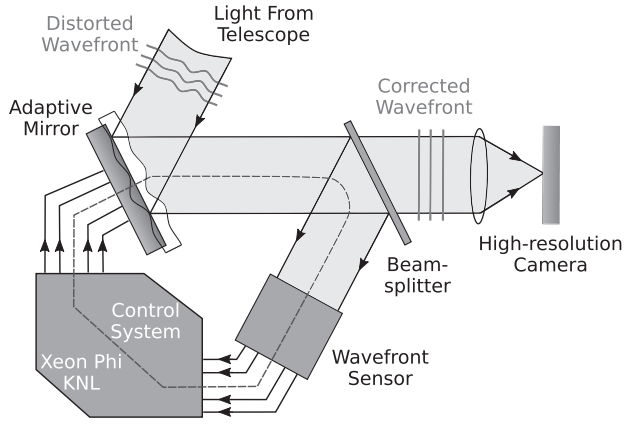

Figure 1. Schematic overview of closed loop adaptive optics. The control system is responsible for taking WFS images, processing them, reconstructing the incident wavefront, and delivering the corrections via the adaptive mirror.

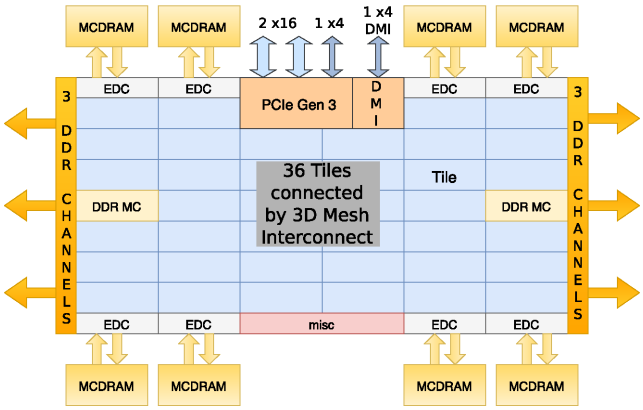

Figure 2. Schematic of Xeon Phi Knights Landing CPU showing the MIC architecture along with the high-bandwidth MCDRAM. Each tile contains two CPU cores, two vector processing units per core, and a shared 1MB of Level2 cache. [DDR MC = DDR memory controller, DMI = Direct Media Interface, EDC = MCDRAM controllers, MCDRAM = Multi-Channel DRAM (Intel 2016)]

change for different types of scientific observation and for different operating wavelengths.

There are different classifications of AO control systems ranging from comparatively simple single conjugate AO (SCAO), which uses a single guide star for correction with a single DM, to more complex systems such as multiple conjugate AO (MCAO, Dicke 1975; Beckers 1988; Johnston & Welsh 1994) and laser tomographic AO (LTAO, Murphy et al. 1991; Tallon & Foy 1990) which use many different reference stars and can employ multiple correcting elements. This paper will concentrate on the SCAO regime.

1.1.1 ELT-scale AO RTC

The dependence of AO system requirements on telescope diameter is an extremely important consideration for the next generation of extremely large telescopes (ELTs), including the Giant Magellan Telescope (GMT, Johns et al. 2004), the Thirty Meter Telescope (TMT, Stepp & Strom 2004), and the European Southern Observatory (ELT, Spyromilio et al. 2008), with primary mirror diameters greater than 20 m. The computational complexity of the conventional wavefront reconstruction algorithm scales with the fourth power of telescope diameter; this is due to the dimensions of the control matrix which are governed by the number of correcting elements in the DM and the number elements of the WFS which both scale to the second power of telescope diameter. This presents a huge challenge in the process of designing an RTC suitable for ELT scale AO, both in the choice of hardware suitable to process the computational demands and with producing software capable of delivering performance that meets the requirements of the AO system.

Many typical AO reconstruction techniques involve computing one or more large matrix-vector-multiplications (MVMs) where the dimensions of the matrix are defined by the number of degrees of freedom (DoF) in the system. A control matrix, which contains information relating to how certain wavefront measurements correspond to the appropriate actuator commands, is multiplied by a vector containing the wavefront slope measurements, the results of which can yield the wavefront shape required for correction. A control matrix of dimensions ($N \times M$) is made up of N DoF of a wavefront slope measurement and M actuators in the correcting element. This makes the MVM very large for ELTs requiring on the order of N^2 calculations where $N \approx M$.

The large MVM calculation is a memory bandwidth bound problem, since the processing time is dependent on the rate that the processing unit can read the large matrix from memory (typically several GB), each AO RTC cycle. This therefore favours computational architectures with large memory bandwidth. The Intel Xeon Phi (Intel 2017) is one such architecture, containing a 16 GB MCDRAM package, with a measured bandwidth as high as 480 GB s^{-1}. Modern graphics processing units (GPUs) also have large memory bandwidths. However, a standard x86 CPU will typically have an achievable memory bandwidth of <90 GB s^{-1} and is therefore at a massive disadvantage for rapid processing of large MVMs. They are therefore not well suited for ELT-scale AO RTC. The Xeon Phi contains a large number of cores, has a large memory bandwidth, and is also programmable as a conventional processor.

1.2 Real-time controller hardware

The first consideration when designing an AO RTC is to choose hardware that can meet the requirements of the AO system, both in terms of input and output (I/O) interfaces for the instruments and in terms of computational performance for the algorithms required. The computational requirements are largely dictated by the reconstruction problem size. Historically, a variety of hardware architectures have been used for AO RTC, including digital signal processors (DSPs, Fedrigo et al. 2006), field programmable gate arrays (FPGAs, Fedrigo et al. 2006; Rodríguez-Ramos et al. 2012), central processing units (CPUs, Basden et al. 2010b), and graphics processing units (GPUs, Basden et al. 2010b; Truong et al. 2008).

These architectures have proved capable for previous and current AO systems with varying advantages and disadvantages for each. The main disadvantage with DSPs, FPGAs, and GPUs is the time cost associated with designing, writing, and, if necessary, modifying the RTC software. The main advantage of FPGAs and DSPs is their deterministic behaviour. Due to their more general computing nature, CPU based systems can be at a disadvantage when it comes to some specific computation problems, such as highly parallelizable problems which may be better suited for GPUs, however, they are much easier to develop for, and are generally backwards compatible with common programming languages. For ELT-scale systems, two of the main challenges are scaling of these systems for the increased

Table 1. Available Knights Landing models and their key specifications. The peak single precision (SP) TFLOPS is a theoretical calculation resulting from the core count, the clock speed (−200 MHz for 512bit vector operation), the vector register size, the number of vector process units per core, and the number of floating point operations per fused multiply-add. e.g for the 7210 model: $64 \times (1.3 - 0.2) \times 512/32 \times 2 \times 2 = 5325$ GFLOPS. The memory bandwidth is that as measured using the STREAM triad benchmark.

KNL Model	Core count	Base CPU clock speed (GHz)	Peak SP TFLOPS	Memory speed (GT s^{-1})	Memory bandwidth (GB s^{-1})
7210	64	1.3	4.51	6.4	450
7230	64	1.3	4.51	7.2	–
7250	68	1.4	5.22	7.2	480
7290	72	1.5	5.99	7.2	–

computational complexity (usually requiring many of these devices working in parallel), and future proofing the development of the system for updated hardware.

The Intel Xeon Phi processor family combines many low power x86 CPU cores utilising wide 512 bit vector registers with high-bandwidth on-chip memory to enable acceleration of highly parallelizable tasks, while keeping the cores sufficiently fed with data. These x86 cores use the backwards compatible x86 instruction sets which are used in the vast majority of Intel and AMD based CPU systems. This allows the Xeon Phi to leverage the benefits both of having a CPU based architecture and of having a highly parallelizable workflow similar to that of GPUs. These attributes of the Xeon Phi make it a very interesting candidate for an ELT-scale AO RTC as they can be developed using conventional CPU programming techniques. However due to the relatively low performance of an individual Xeon Phi CPU core, properly utilising vectorization and parallelism is essential for good performance.

In this paper we present an investigation of the suitability of the Intel Xeon Phi Many Integrated Core (MIC) architecture as the processor for ELT-scale AO RTC. Section 2 summarizes our efforts with adapting existing RTC software for use with the Xeon Phi platform, Section 3 presents our results, and Section 4 concludes with a summary and a discussion of our work.

1.2.1 Xeon Phi: Knights Landing

Knights Landing (KNL) is the third generation Xeon Phi processor and is the first to be released in the self-booting socketed form factor, with a number of variants, as given in Table 1. Previous generation Xeon Phi chips were available as accelerators only. The KNL processor can therefore be used just like a conventional server processor and can run the Linux operating system and standard software environment. Existing applications can be ported to the Xeon Phi quickly: recompilation is usually not even required, though code will not be well optimized in this case.

The KNL introduces additional instruction sets, which can be utilized for improved performance, including 512 bit CPU registers, which allow single instruction multiple data (SIMD) operation on 16 single-precision floating point numbers simultaneously per core, per instruction cycle. This improves the parallelization advantage of the KNL architecture over previous processors, which included a maximum of 256 bit wide vector registers. This 512 bit register is also included in forthcoming (and most recent) standard Intel CPUs, so any code optimizations made for this feature will also be applicable to future non-Xeon-Phi CPUs. However, it should be noted that for the KNL system, high utilization of 512 bit instructions reduces the base core clock speed by 200 Hz, which is taken into consideration in the peak SP TFLOPS calculated in Table 1.

The wavefront sensor cameras can be directly attached to the KNL via the PCIe bus. This is an advantage over accelerator cards such as previous the generation Xeon Phi cards and GPUs, where, unless specific effort is taken (often requiring specific network cards and low-level device programming), image data must be transferred first to the CPU from the camera and then out to the accelerator (essentially 2 PCIe transfers with increased latency and jitter). The previous generation of the Xeon Phi (Knights Corner) has also been investigated for AO RTC (Barr et al. 2015), showing promise for future processor generations (i.e. the KNL).

2 DEVELOPMENT OF AO RTC USING A XEON PHI

A large component of the time and effort required to design and produce an AO RTC stems from development of the control software. For technologies such as DSPs, FPGAs, and GPUs, this can be extremely time consuming and require specific technological expertise without any guarantee that the software will be in any way compatible with future devices. CPU programme development is comparatively more straightforward with a choice of well documented and easily accessible programming languages to choose from which are compatible with a wide range of CPU based platforms.

2.1 Best case performance for ELT-scale SCAO RTC

In order to determine the best performance achievable with a Xeon Phi, we have developed a highly optimized algorithmic RTC, i.e. a simple software solution which performs all the necessary RTC algorithms using optimized library functions, but which is not pipelined, does not interact with real camera or DM hardware, and is not user configurable. Therefore, although this RTC cannot be used in a real AO system, it gives some idea of the minimum frame computation time (or maximum frame rate), which can be achieved using this hardware. We note that an investigation using a full on-sky tested RTC is introduced in later sections.

The simple simulator uses the OpenMP API (OpenMP Architecture Review Board 2015) for multithreading and performs pixel calibration on fake image data, centroiding of the calibrated pixels, an MVM reconstruction of the centroids, and finally introduces a gain factor to the final result. The slope measurements are computed as if all pixels are available at once. This is the minimum computational requirement of an SCAO RTC and gives a base-line for best case expected performance of the Xeon Phi. Fig.3 gives an overview of this system.

2.2 Real RTC software using the Durham AO real-time controller

We have optimized a real on-sky tested RTC for the Xeon Phi KNL in the form of the Durham adaptive optics real time controller (DARC, Basden et al. 2010b). DARC is a freely available and on-sky proven AO RTC software package written in the C and PYTHON programming languages. It is built upon a modular real-time core which allows it to be extended for many different AO RTC scenarios such as for different AO regimes like SCAO and MOAO and allows individual algorithms such as pixel calibration and wavefront reconstruction to be replaced or modified. The modular design also allows it to interface with many different devices for wavefront sensing and

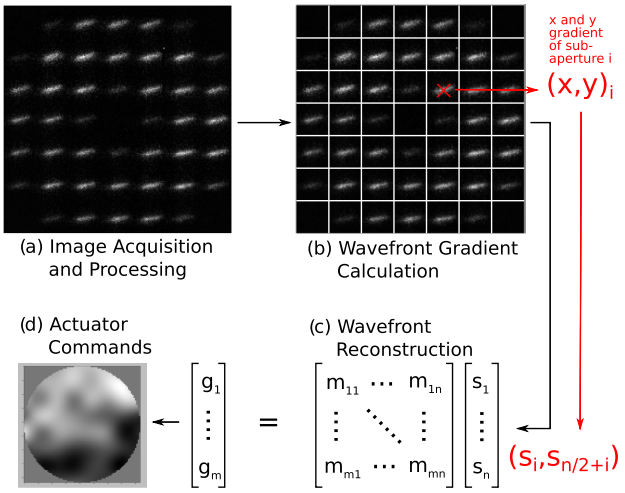

Figure 3. A figure showing the basic RTC operations, including a) image acquisition and processing (background subtraction, flat field application, and threshold application), b) local wavefront gradient computation (using a centre of gravity algorithm), c) wavefront reconstruction (using an MVM), and d) output of actuator commands. A thread will process a defined set of subapertures from beginning to end. For each subaperture, the local wavefront gradients are placed in a slope vector such that all the x gradients come first and then the y gradients $[(x, y)_i -> (s_i, s_{n/2+i})]$. The result of the MVM is a vector of actuator commands which can be reformatted to show the resulting shape of the correcting element.

wavefront correction (Basden et al. 2016), making it flexible enough to be used in almost any AO situation. Because DARC is built on the C and PYTHON programming languages it can be compiled and run on many different systems including the socketed Xeon Phi, as well as x86, POWER8 (Basden 2015), and ARM processors. Within DARC, wavefront sensor images are processed in parallel, with subapertures being processed as soon as enough pixels have arrived at the computer. DARC uses a horizontal processing strategy which allocates a similar workload to each thread, with threads being responsible for processing of a subaperture from start to finish (including calibration, slope calculation and partial reconstruction). This means that AO latency can be reduced, since by the time the last pixels arrive at the computer, the majority of the processing for that frame has already been completed. Here, we consider the optimization of DARC for use with the Xeon Phi architecture, and report on performance investigations.

2.3 Main areas of DARC optimization for the Xeon Phi

As an x86 CPU the Xeon Phi shares many attributes with standard CPU hardware. However, it is also very different from previous CPUs with its many (≥64) low-power cores, its high-bandwidth memory, and the 512 bit wide vector registers for improved SIMD performance. While most software developed for standard CPU systems can be compiled and run on Xeon Phi hardware with no alterations, to make the most of the new features, some optimizations are needed to best utilize the available hardware, these include:

(i) thread synchronization, to make efficient use of all cores
(ii) memory access, to optimize for the fast memory
(iii) vectorization, to take advantage of the wide vector registers.

2.3.1 BIOS, operating system, and kernel setup

The operating system (OS) installed on the Xeon Phi used in this paper is CentOS Linux 7.3 (The-CentOS-Project 2001). To obtain the best low-latency and low-jitter performance various changes have been made to the default settings of the BIOS, the operating system, and the kernel. The main changes to the BIOS settings involve turning off Intel hyper-threading, which allows more logical threads to execute concurrently on hardware cores. Removing Hyper-threading allows each software thread to be pinned to a single hardware core and removes scheduling inefficiencies caused when cores switch between different hyper-threads.

Other BIOS settings include Xeon Phi specific settings which relate to how the CPU handles memory addressing, with information available online (Intel 2015), and different modes which determine how the fast multichannel DRAM (MCDRAM) is allocated, either accessible like standard RAM, reserved for the OS as a large last level cache (LLC), or a mixture of the two; these modes are termed 'flat', 'cache', and 'hybrid' respectively.

OS and kernel setup refers to options such as isolating certain CPU cores so that the OS doesn't schedule any programme to run on these cores without specific instruction, and other options relating to CPU interrupts and different power and performance modes.

During our testing, we have identified that best performance is achieved with the CPU set to quadrant memory addressing mode, and the MCDRAM was set to 'flat' mode. In 'flat' mode, the MCDRAM is visible to the CPU on a separate NUMA node from the standard RAM and so this must be addressed either by explicitly allocating the memory in the programme (using a NUMA library), or by executing the programme on the specific NUMA node to make use of the fast MCDRAM. In this report the MCDRAM was allocated by running software with the 'numactl' command with the '–membind=nodes' option, ensuring that the entire RTC is allocated on this NUMA node. On the Xeon Phi, the MCDRAM is 16 GB in size, which is sufficient to fit a whole ELT-scale RTC.

2.3.2 Multithreading and synchronization

Multicore CPU systems have become the norm in recent years leading to DARC being developed using a multithreaded real-time core with the POSIX ('The-Open-Group' 2016) pthread library. The main method of ensuring thread synchronization has been by the use of pthread mutexes and condition variables. A mutex is a mutual-exclusion variable which allows threads to 'lock' a certain section of code, preventing other threads from accessing these protected regions. If a thread calls the lock function on an unlocked mutex variable, then that thread will be allowed to proceed and then the mutex becomes locked. Any other threads which attempt to lock this mutex will have to wait at the lock function until the mutex is unlocked.

If multiple threads are waiting at a mutex then they will proceed one by one as the mutex is repeatedly locked and unlocked by the preceding thread. A thread waiting at a mutex will generally be descheduled by the operating system scheduler and put to sleep, reducing power consumption and freeing up the hardware for other threads to be scheduled. This works well for low-order multicore systems with 2–16 CPU cores, as it allows for more threads than physical CPU cores and the simultaneous descheduling and rescheduling of these few threads when they are waiting at the same mutex has little overhead.

However, for the Xeon Phi MIC architecture with ≥64 low-power cores, DARC needs to be configured to execute a single thread per core

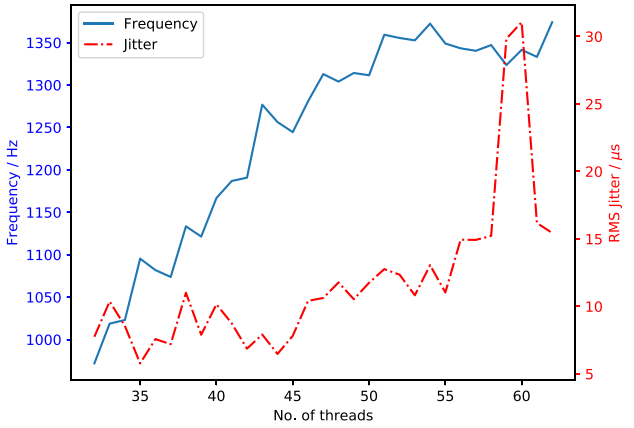

Figure 4. A measure of the average frequency of DARC running with different numbers of threads on Xeon Phi KNL, given in Hz. Also shown is the RMS jitter of the frametime data used given in µs. The periodic structure seen at higher thread counts is most likely due the thread allocation used to aid in vectorization described in Section 2.3.3.

with >48 threads to achieve maximum performance for ELT-scale SCAO (Fig. 4). This can cause problems when using mutexes and condition variables as the constant sleeping and waking of this large number of threads significantly increases latency. The solution that we have developed is to use a structure similar to mutexes called spinlocks, which also protect critical sections of parallel code but instead of sleeping and descheduling, threads simply wait until they can proceed. This waiting process constantly consumes CPU cycles but increases the system's responsiveness which massively reduces the latency when using many threads as it is then much faster to resume operation.

Unfortunately, condition variables do not work with spinlocks and so we replace these where possible by simple volatile flag variables, taking care to ensure that thread safety is maintained.

2.3.3 *Vectorization*

The 512 bit wide vector registers present on the Xeon Phi allow up to 16 single precision (SP) operations to be performed per cycle per CPU core. An operation in this case can be a fused-multiply add (FMA) operation which combines an addition and a multiplication, allowing up to 16 SP additions and 16 SP multiplications per instruction cycle. This is double the previous specification of 256 bit vector registers allowing a theoretical 2X speed up for vectorizable computations. Vectorization is generally handled by the compiler: depending on the level of optimization chosen at compile time, a certain amount of autovectorization will occur. However, steps can be taken to aid the compiler and investigate where vectorization occurs or does not occur. Essentially, if the compiler is able to detect that vector or matrix operations include 16 float boundaries at the same points, then these operations can be vectorized. This therefore usually means that by aligning memory to the nearest 64 bytes, vectorization will be aided.

The allocation of subapertures to specific threads within DARC can be optimized such that each thread processes a multiple of 16 slope measurements when calculating its own section of the wavefront reconstruction MVM. As each subaperture has two slope measurements, one for each of the x and y directions, we therefore ensure that the subapertures are allocated to threads such that each thread processes a multiple of eight subapertures as a block.

Alignment of array memory to page cache boundaries is important so that the data required for the vectorized instructions can be loaded into the registers efficiently and with the right ordering. This can be done when allocating memory for the arrays using the posix_memalign ('The-Open-Group' 2016) function call which aligns the amount of memory required at the specified boundary. The next step is to then ensure that sections which can be vectorized are written in such a way that the compiler can apply autovectorization; Intel provides a guide which details the necessary steps (Intel 2012).

2.3.4 *Reducing the memory bandwidth requirement*

Because the wavefront reconstruction MVM is memory bandwidth bound, due to the relatively simple mathematical operations but large data size, investigating ways to reduce the memory bandwidth dependence is an important consideration for ELT-scale AO RTC. A potential solution is to store the control matrix using 16 bit floating point format, rather than the conventional 32 bit format. The format used for the 16 bit floats is the IEEE 754 specification for binary16 (IEEE 2008) which reduces the exponent from 8 to 5 bits and the mantissa from 23 to 10 bits.

This change does result in some loss of precision in the control matrix, however, the available precision is still greater than that of the wavefront slope measurements (which are based on integer-valued detector measurements), and is therefore still considered sufficient for the reconstruction (Basden, Myers & Butterley 2010a). Every AO loop iteration, this control matrix is then loaded into CPU registers, converted to 32 bit format for operations (necessary since the Xeon Phi cannot perform 16 bit floating point mathematical operations), and the DM vector computed. The reduction in memory bandwidth required can therefore reduce AO system latency.

2.3.5 *Optimizing parallel vector-addition*

As each DARC thread processes a set of subapertures from beginning to end, the result of each thread's execution is a partial DM vector for those subapertures. To combine these results into a final DM command vector they must be all be summed together. Previously DARC has achieved this by using a mutex to lock the final DM command vector whilst each thread adds its vector into it in turn. This works well for small numbers of threads on fast cores, however, for the KNL case where there are more threads on comparatively slower cores, this serial addition can be a source of a large amount of extra latency.

A solution that we have developed is a branching tree-like algorithm which allows groups of threads to add their partial DM vectors and so each group can work in parallel. A group will be defined by a spinlock and a thread-barrier. Within the group a thread will get the lock whilst it copies its partial DM vector into a temporary output array and once each thread has copied its partial vector, it waits at the barrier for the other threads in the group to finish. At the completion of a group's work, one thread from each group will take ownership of the temporary output array and move on to the next group. This can be seen in Fig. 5, where each group adds its partial DM vector into the red box before that moves down to the next group where the process is repeated until the final DM command vector results.

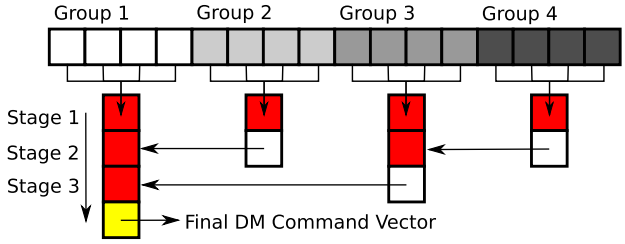

Figure 5. A schematic of the branching tree-like vector addition algorithm for a $4 \rightarrow 2 \rightarrow 2$ situation with 16 threads, the first stage involves groups of four threads adding up their partial DM vectors, stages 2 and 3 reduce the resulting temporary DM commands to final DM command vector. This example allows up to four vector additions to happen in parallel and a total of three sequential stages instead of simply adding up all 16 threads' partial DM vectors sequentially. For larger thread counts, the effect is even more pronounced.

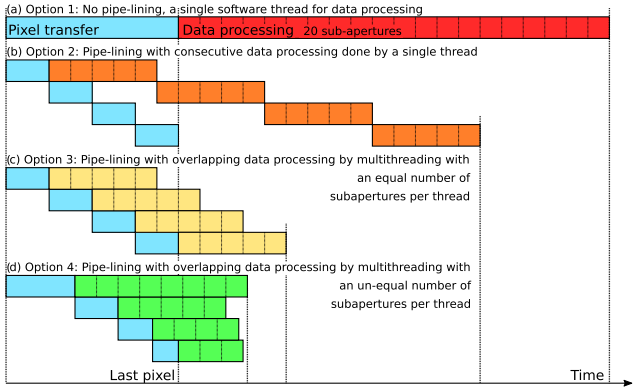

Figure 6. A comparison of the latency introduced via various pixel handling techniques. It shows that minimal latency is achieved via pipelining of the reconstruction using threads which process unequal numbers of subapertures such that they finish processing at roughly the same time.

2.3.6 Reducing RTC using latency with asymmetric subaperture thread allocation

The latency of an RTC is defined as the time from last pixel to readying the final DM command and so reducing this time interval is paramount to improving the performance of the RTC. The different options for handling and processing the pixel stream are shown in Fig. 6, with each subsequent option reducing the time taken from last pixel to end of thread computation.

As each DARC thread processes its subapertures from beginning to end, they must be allocated a specific set of subapertures to process, the most simple and naive way of assigning subapertures would be to divide them equally amongst the threads, see Fig. 6(c). However as can be seen the threads which process the earlier arriving subapertures finish their work before the later threads, the subapertures are assigned equally among threads in an ascending order, as the thread number increases so does the time waiting for pixels, yet the processing time is constant.

Seen in Fig. 6(d) is an option whereby the subapertures are not allocated equally among the threads, the threads that process earlier subapertures are given more work to do and the later ones are given less. This ensures that the threads finish their work at roughly the same time and so helps to reduce the time between the last pixel arriving and the final DM command being sent out. However, the time waiting for pixels also changes as processing more subapertures requires waiting for more pixels, which can be seen in the different sized blocks for pixel waiting. This is a lot more complex in practice as there will not be a one-to-one relation between the number of subapertures and the number of pixels that need to be waited for.

3 RESULTS AND DISCUSSION

The performance of an AO RTC is generally defined by the time taken for it to process each frame, where a frame is a single image from a WFS and the processing involves calibrating the pixels, computing the centroids and reconstructing the wavefront before sending the results to a correcting element. There are different ways of determining the amount of time that it takes an RTC to process a frame and it depends on the definition of when a frame starts and when it ends. Timing for the entire AO RTC loop is generally taken from the time the first pixels arrive at the RTC core to the time when the last DM commands have been sent to the correcting element.

This encompasses the entire computation of the RTC especially when the main loop is pipelined, i.e., the processing is done for groups of pixels as they arrive due to the way the image sensors read-out the pixels. However, in the case of a fast RTC and a slower camera, RTC latency will be artificially lengthened by periods waiting for camera pixels to arrive. Here, we use the traditional definition of RTC latency, defined by the time taken between last camera pixels arriving at the RTC and last DM demand being computed. This definition can therefore be computed entirely within the RTC hardware, though does not include delays due to the capture of camera pixels (e.g. by a frame grabber card, and transfer to the computer memory), or delays due to time taken for DM demands to leave the computer and arrive at the DM. In addition to latency, we also report maximum stable RTC loop rate, i.e. the fastest rate at which the RTC can operate stably.

In cases where we present maximum RTC rate, i.e. without a camera attached, we define latency as the inverse frametime: in this case, the latency represents the minimum computation time for the RTC loop.

We also define the jitter of the AO RTC to be the variation in latency. We present both rms jitter and also peak-to-peak (worst case) jitter.

3.1 Suitability of Xeon Phi for ELT scale AO RTC

We configure the best case SCAO RTC simulator in an ELT configuration with 80×80 subapertures with a $0.25 \times D$ central obscuration and 10×10 pixels per subaperture. This results in 4708 active subapertures, and therefore 9416 slope measurements. The number of DM actuators is 5170, based on a circular 81×81 actuator DM aperture.

Fig. 7 shows the frametime results of the SCAO best case simulator for 10^6 frames. The figure shows the minimal number of outliers and also the small spread of the distribution. The average frametime of 0.77 ± 0.02 ms corresponds to an average framerate of 1300 ± 30 Hz. This is shorter than a typical atmospheric coherence time, and therefore would be suitable for ELT-scale SCAO RTC. The rms jitter is 16.3 μs, which is about 2 per cent of the mean frametime, and would have insignificant impact on AO performance (Pettazzi, Fedrigo & Clare 2012). The maximum instantaneous peak-to-peak jitter between consecutive frames is 107 μs, including the startup measurements, or 88.8 μs during the long-term measurements.

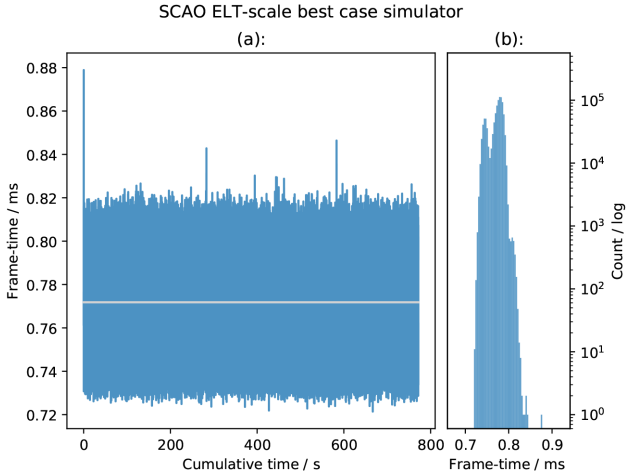

Figure 7. a) Frametime results of SCAO best case simulator for 10^6 frames with an average frametime of 0.77 ± 0.02 ms which corresponds to an average framerate of 1300 ± 30 Hz. The horizontal line is the average frametime. b) A histogram of the frametimes in (a).

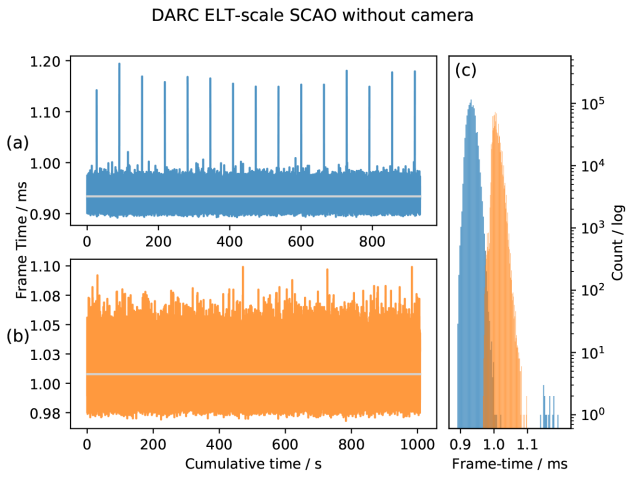

Figure 8. a) Frametimes for DARC SCAO on an Intel motherboard with no camera for 10^6 frames with an average frametime of 0.93 ± 0.01 ms which corresponds to an average framerate of 1070 ± 10 Hz. b) As for (a), except for a Supermicro motherboard with an average frametime of 1.01 ± 0.01 ms which corresponds to an average framerate of 990 ± 10 Hz. c) Histograms of the frametimes presented in (a) and (b) with (a) on the left and (b) on the right. The horizontal lines in (a) and (b) are the mean frametimes for each distribution.

3.1.1 DARC on Xeon Phi for ELT scale AO RTC

Fig. 8 shows the frametime results of DARC configured for ELT-scale SCAO (in a similar configuration as above, however, with 5316 actuators, to mimic the ELTs M4 Adaptive mirror, and 4632 subapertures for a total of 9264 slope measurements) though without a physical camera connected, measured over 10^6 frames. It can be seen in Fig. 8(a) that for the system using an Intel motherboard (model S72000, 7250 processor), there are a small number of regular single frame outliers which add about 200–250 μs to the frametime, roughly every 63.75 s. We have determined that these events are due to the Intel system management interface on the motherboard, which periodically polls the processor for information. There appears to be no way in which this can be turned off. The presence of these interrupts can be verified using this code:

(i) For SEC in ‘seq 0 200‘; do echo -n ‘$SEC ‘; rdmsr -p 0 -d 0x34; sleep 1; done,

which has been used to confirm their presence on the Intel S72000 motherboard used in this report.

Fig. 8(b) shows results taken using a Ninja Development platform Xeon Phi using a Supermicro motherboard (model K1SPE with 7210 processor). Here it can be seen that these 64 s period events are not present. It is therefore important to take care when evaluating motherboards suitable for AO RTC. histograms of both measurements are shown in Fig. 8(c), the difference in mean frametime between the two distributions is due to the specification of the processors used in each motherboard; 1.4 versus 1.3 GHz clock speed, 480–450 GB s^{-1} memory bandwidth (Table 1). From this figure, it can be seen that the distribution of latency measurements is approximately Gaussian, except for the outliers.

Therefore, DARC is able to operate ELT-scale SCAO with a 0.93 ± 0.01 ms frametime, corresponding to a 1070 ± 10 Hz maximum frame rate. When the 64 s events are included, the instantaneous peak-to-peak jitter over a million frames is 263 μ s, while ignoring these events reduces the peak-to-peak jitter to 92.7 μ s and the RMS jitter is only 11.4 μs. The Ninja development platform can operate ELT-scale SCAO with a 1.01 ± 0.01 ms frametime, corresponding to a 990 ± 10 Hz maximum frame rate. This is a lower maximum performance than the Intel motherboard system due to the difference in processor specification which is as expected.

3.1.2 Storing the control matrix as 16 bit floating point values

For investigating the effect on the latency of the RTC on the Xeon Phi when storing the control matrix as 16 bit floating point values we had to use our own implementation of an MVM algorithm. This is because the Intel MKL library that is used to calculate the reconstruction MVM in previous results is unable to operate directly on 16 bit floating point values. To be able to use it, the values would need to be converted to 32 bit before each call to the MKL library. This would not be ideal as MKL works best on larger MVM problem sizes and converting a large amount of the control matrix to 32 bit would defeat the purpose of storing it as 16 bit and making too many calls to the library would vastly increase the latency.

Our custom MVM implementation uses Intel intrinsic instructions to convert the 16 bit values of the control matrix immediately before they are operated upon. Up to 16 conversions can be done in a single instruction with the results stored in one of the 512 bit vector registers ready for the FMA MVM operations. This ensures that there is a minimum transfer of 32 bit values to conserve memory bandwidth.

A similar custom 32 bit MVM implementation, simply loading the data instead of converting it, shows that this algorithm is not as optimized as MKL. It gives an average frametime of 0.995 ± 0.006 ms with an RMS jitter of 5.96μs, which can be compared to results that use MKL on the same processor/motherboard of 0.93 ± 0.01 ms from Fig. 8.

The algorithm that uses a 16 bit control matrix decreases this average frametime to 0.902 ± 0.006 ms with an RMS jitter of 5.60 μs. The need to use a less optimized custom MVM and the need to convert to 32 bit floats introduces extra overhead which greatly reduces the potential gain. These results show that this implementation of storing and converting the 16 bit control matrix increases performance by only 3 per cent over the best case MKL results.

The next iteration of Xeon Phi after KNL, Knights Mill (KNM) which is available now, includes support for Intel variable preci-

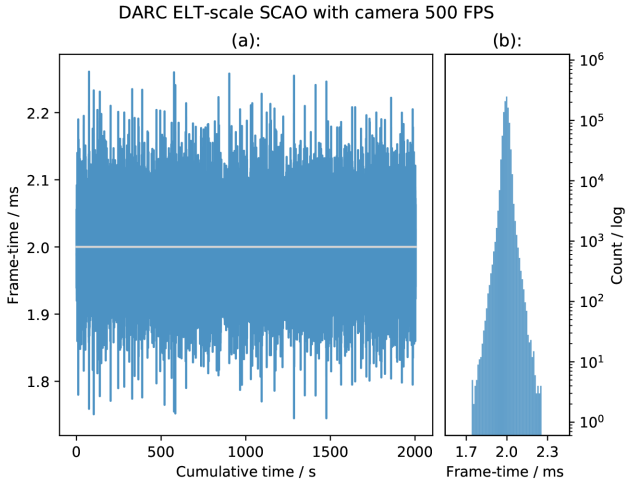

Figure 9. a) Frametimes of DARC SCAO with a real wavefront sensor camera operating at 500 Hz for 10^6 frames, the horizontal line shows the average frametime of 2.00 ± 0.02 ms. b) A histogram of the frametimes, showing the distribution of jitter.

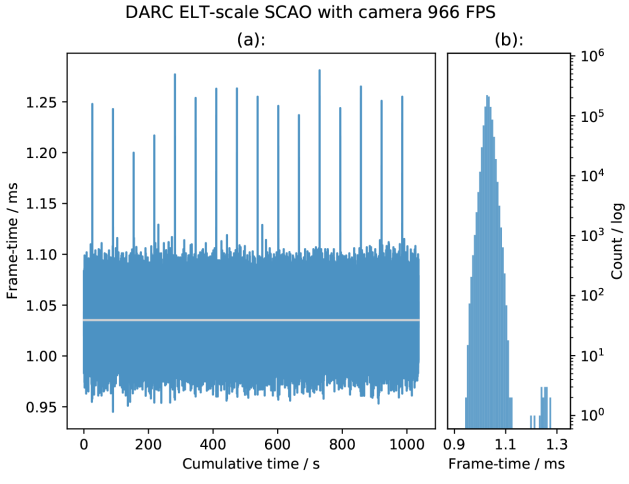

Figure 10. a) Frametimes of DARC SCAO with a real wavefront sensor camera operating at 966 Hz for 10^6 frames, the horizontal line shows the average frametime of 1.04 ± 0.01 ms. b) A histogram of the frametimes, showing the distribution of jitter.

sion operations (vector neural network instruction, VNNI) which include intrinsic instructions that can directly operate on 16 bit integer values by addition and multiplication to produce an accumulated 32 bit integer sum. This would require some conversion of the control matrix and slope values to fixed-point 16 bit precision integers but could reduce the memory bandwidth requirement without introducing a costly 16 to 32 bit conversion for each control matrix value at execution time. Simulations have shown that 16 bit fixed-point values would just be sufficient to provide the required precision (Basden et al. 2010a), however, when taking into account a real system with misalignments it may not be adequate.

This functionality comes via a redesigned vector processing unit (VPU) which also doubles the number of SP FMA operations possible per instruction cycle, increasing the theoretical peak SP-FLOPS 2X over KNL. This is achieved with quad FMA (QFMA) instructions which allow for sequential FMA to accumulate over four sets of calculations with a single instruction. There are caveats to this, however, due the instruction pipeline of the VPUs, so 2X speed up is unlikely but the QFMA operations should definitely benefit the highly vectorizable MVM without needing to use the 16 bit VNNI; investigation into KNM VNNI and QFMA instructions could be useful for future work.

3.2 DARC SCAO computation with a real wavefront sensor camera

Figs 9 and 10 show two sets of frametime results for DARC configured for ELT-scale SCAO, with pixels arriving from a real 10 GigE Vision based camera running at 500 Hz and at the camera's maximum framerate of 966 Hz respectively. The camera is an Emergent Vision Technologies HS2000M, delivering 100 pixels per subaperture. The figures show minimal numbers of outliers and also the small spread of the distributions, with rms jitters of 20.1 µs and 13.8 µs for 500 Hz and 966 Hz respectively, which is similar to that when DARC operates without a real camera.

The 64 s events due to the Intel motherboard are visible in the data for 966 Hz, however, they are not seen in the data for 500 Hz, this is likely due to the reduced computational demands for SCAO at 500 Hz and so the CPU has ample time to process the interrupts without affecting DARC. The maximum instantaneous peak-to-peak jitter is 510µ s for 500 Hz and 163 µ s for 966 Hz excluding the 64 s events, over one million frames.

The shapes of the histograms in Figs 9 and 10 are quite different, as they are plotted on a log scale it shows that for the camera operating at 500 Hz there is a high narrow peak at the mean of the distribution with relatively small numbers of frames spread out to either side. This gives the distribution its low RMS jitter but a relatively high instantaneous peak-to-peak jitter.

We use a modified version of the Aravis GigE Vision library (AravisProject 2018), which enables access to the pixel stream, rather than waiting until the entire frame has been delivered. In this way, DARC can begin processing subapertures as soon as enough pixels have arrived, reducing latency. As the latency of an RTC is defined as the time between last pixel arriving and the final DM command being sent out, reducing this time improves the performance of the RTC.

Fig. 11 shows the time taken for the DARC processing threads to finish processing their subapertures from the time the last pixel arrives from a real camera; these are average times for 10^5 frames. Fig. 11 (Equal number) is for the case described in Figs 6(c) and 11 (Unequal number) is for the Fig. 6(d) case. Both sets of data are taken with the real camera operating at 500 Hz. Fig. 12 shows that for the situation with equal numbers of subapertures the mean RTC latency for 10^5 frames is 0.84 ± 0.02 ms, and for unequal numbers, the mean RTC latency is 0.64 ± 0.02 ms. Fig. 11 shows a modest improvement in RTC latency, bringing the latency below that of the best case simulator and demonstrates the different end of thread execution times described in Fig. 6. These results show that DARC on the Xeon Phi can operate SCAO at ELT-scales with a real camera.

The algorithm used to assign the unequal numbers of subapertures is a very basic implementation with a linearly decreasing subaperture count per thread. This algorithm will be explored further to find the optimal subaperture allocation to improve latency for desired framerates and different read-out rates.

3.3 Best case and DARC performance

The results described in Sections 3.1.1 and 3.2 for the mature DARC AO RTC on the Xeon Phi are consistent with those found in Section 3.1. The frametimes for DARC with no camera, as shown in

Figure 11. A measure of the time from the last pixel arriving from the camera to end of thread computation per thread. 'Equal number' shows the results for a naive subaperture allocation whereby each thread processes an equal number of subapertures. 'Unequal number' shows allocation by a simple algorithm which gives more work to threads which are processing subapertures whose pixels arrive earlier.

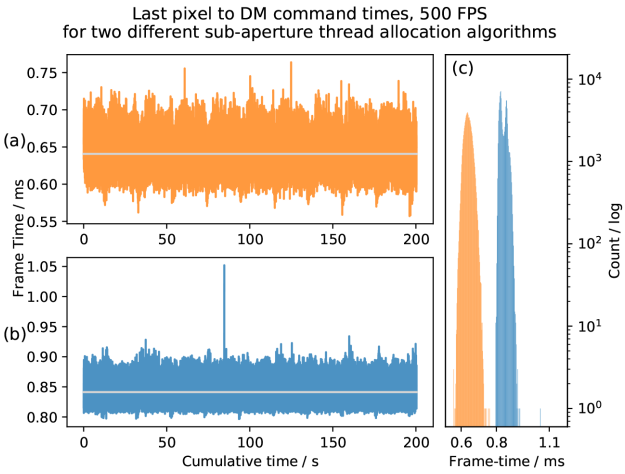

Figure 12. Latency measurements (time between last pixel received to DM demand ready) for DARC operation with a real wavefront sensor camera at ELT-scale at 500 Hz. a) shows results when using the unequal subaperture thread allocation. b) shows results for the equal subaperture thread allocation, see Fig. 11.

Fig. 8, give the current best performance of the computation along with pipelining and hard real-time aspects which are required for a real RTC. DARC operates with a maximum frame rate which is slightly below that of the best case system, though will also exhibit lower latency, due to the ability to interleave processing and pixel acquisition (which the base-case system cannot do). Fig. 12 shows that by using a more appropriate subaperture thread allocation algorithm, the latency of the RTC with a real camera can be reduced to well below that of the best case simulator.

3.4 Future work

The optimizations of DARC presented in this paper are far from exhaustive, they focus on the multithreading and reconstruction aspects and provide us with adequate performance for ELT SCAO. However, they can still be improved upon by further investigation into other aspects of the real-time pipeline such as the pixel calibration and centroiding parts. There are also more novel algorithms in use in AO which could benefit from acceleration by Xeon Phi such as more complex reconstruction techniques like linear quadratic Gaussian (LQG, Kulcsár et al. 2006) reconstruction and less computationally expensive techniques such as the cumulative reconstructor with domain decomposition (CuRe-D, Rosensteiner 2012).

Another area where Xeon Phis may be applicable is for ELT scale AO modelling and simulation, to aid design decisions during system design studies. Currently full ELT scale simulations are not capable of being run even close to real-time (Basden 2014), and therefore the large AO parameter space takes a long time to fully explore. To aid with ELT commissioning, a hardware-in-the-loop real-time simulation (Basden 2014) will be required, and here, operation at close to real-time rates becomes very important. AO PSF reconstruction will also benefit from real-time simulation rates.

Investigation into the improved 16 bit integer instructions, such as VNNI and QFMA described in Section 3.1.2, provided by the Xeon Phi Knights Mill could also be considered for future investigation. Hardware that supports half-precision integer instructions are becoming ever more common with their use in neural network applications and so their investigation for AO RTC would be very useful for future hardware.

3.4.1 ELT MCAO prototype

This paper has only discussed the SCAO regime of AO for ELT-scale but the next generation telescopes all have more complex multiconjugate or multi-object AO systems planned for which suitable RTCs will be required. One of these systems is the ELTs multiconjugate adaptive optics relay (MAORY) which will employ six laser guide stars (LGS) along with three natural guide stars (NGS) using three DMs for the correction, increasing the computation requirements by more than six times that of ELT SCAO. Our initial investigation (Jenkins, Basden & Myers 2018) for this involves multiple Xeon Phi systems working in parallel to process the images from each WFS independently before combining the partial DM vectors and delivering to the DMs. This presents significant challenges in addition to those for preparing DARC for ELT-scale SCAO.

4 CONCLUSION

The results presented show that the Intel Xeon Phi is capable of performing the computational requirements of a full on-sky tested AO RTC for ELT scale with frame-rates and latencies which are suitable for AO system operation. Furthermore, it has been shown that with a real wavefront sensor camera, the RTC is capable of performing at similar framerates albeit with slightly increased jitter. We can recommend the Xeon Phi as suitable hardware for ELT-scale real-time control, primarily due to the large number of cores and the high memory bandwidth, though note that it is necessary to take care when selecting a suitable motherboard, to avoid periodic temporary latency increases.

ACKNOWLEDGEMENTS

Real-time adaptive optics work by the Durham group is supported by the European Union Horizon 2020 funded GreenFlash project,

Identification 671662, under FETHPC-1-2014, the UK Science and Technology Facilities Council consolidated grant ST/P000541/1, and a Science and Technology Facilities Council PhD studentship, award reference 1628730.

REFERENCES

'The-Open-Group', 2016, The Open Group Base Specifications Issue 7 http://pubs.opengroup.org/onlinepubs/9699919799.2016edition/
AravisProject, 2018, Aravis https://github.com/AravisProject/aravis
Babcock H. W., 1953, PASP, 65, 229
Barr D., Basden A., Dipper N., Schwartz N., 2015, MNRAS, 453, 3222
Basden A., 2014, MNRAS, 439, 2854
Basden A. G., 2015, MNRAS, 452, 1694
Basden A. G., Myers R., Butterley T., 2010a, Appl. Opt., 49, G1
Basden A., Geng D., Myers R., Younger E., 2010b, Appl. Opt., 49, 6354
Basden A. G. et al., 2016, MNRAS, 459, 1350
Beckers J. M., 1988, in Ulrich M.-H., eds, European Southern Observatory Conference and Workshop Proceedings, European Southern Observatory, Garching, p. 693
Dicke R. H., 1975, ApJ, 198, 605
Fedrigo E., Donaldson R., Soenke C., Myers R., Goodsell S., Geng D., Saunter C., Dipper N., 2006, in Ellerbroek B. L., Bonaccini Calia D., eds, Proc. SPIE Conf. Ser. Vol. 6272, Advances in Adaptive Optics II. SPIE, Bellingham. p. 627210
IEEE, 2008, IEEE Standard for Floating-Point Arithmetic
Intel, 2012, A Guide to Auto-vectorization with Intel C++ Compilers https://software.intel.com/en-us/articles/a-guide-to-auto-vectorization-with-intel-c-compilers
Intel, 2015, Memory Modes and Cluster Modes: Configuration and Use Cases https://software.intel.com/en-us/articles/intel-xeon-phi-x200-processor-memory-modes-and-cluster-modes-configuration-and-use-cases
Intel 2016, Intel Xeon Phi Processor 7200 Family Memory Management Optimizations https://software.intel.com/en-us/articles/intel-xeon-phi-processor-7200-family-memory-management-optimizations
Intel, 2017, Intelxeon Phi Processors https://www.intel.com/content/www/us/en/products/processors/xeon-phi/xeon-phi-processors.html
Jenkins D., Basden A., Myers R., 2018, in preparation
Johns M., Angel R., Shectman S., Bernstein R., Fabricant D., McCarthy P., Phillips M., 2004, in Oschmann J. M., Jr., eds, Proc. SPIE Conf. Ser. Vol. 5489, Ground-based Telescopes. SPIE, Bellingham, p. 5489
Johnston D. C., Welsh B. M., 1994, J. Opt. Soc. Am. A, 11, 394
Kulcsár C., Raynaud H.-F., Petit C., Conan J.-M., de Lesegno P. V., 2006, Opt. Express, 14, 7464
Murphy D. V., Primmerman C. A., Zollars B. G., Barclay H. T., 1991, Opt. Lett., 16, 1797
OpenMP Architecture Review Board, 2015, OpenMP Application Program Interface Version 4.5 http://www.openmp.org/wp-content/uploads/openmp-4.5.pdf
Pettazzi L., Fedrigo E., Clare R., 2012, Impact of Latency and Jitter on the Performance of Adaptive Optics Systems for ELTs https://www.eso.org/sci/meetings/2012/RTCWorkshop/1_2_fedrigo.pdf
Rodríguez-Ramos L. F., Chulani H., Martín Y., Dorta T., Alonso A., Fuensalida J. J., 2012, in Ellerbroek B., Marchetti E., Veran J-P., eds, Proc. SPIE Conf. Ser. Vol. 8447, Adaptive Optics Systems III. SPIE, Bellingham, p. 84472R
Rosensteiner M., 2012, J. Opt. Soc. Am. A, 29, 2328
Spyromilio J., Comerón F., D'Odorico S., Kissler-Patig M., Gilmozzi R., 2008, Messenger, 133, 2
Stepp L. M., Strom S. E., 2004, in Ardeberg A. L., Andersen T., eds, Proc. SPIE Conf. Ser. Vol. 5382, Second Backaskog Workshop on Extremely Large Telescopes. SPIE, Bellingham, p. 67
Tallon M., Foy R., 1990, A&A, 235, 549
The-CentOS-Project, 2001, CentOS Linux https://www.centos.org/about/
Truong T., et al., 2008, in Hubin N., Max C., Wizinowich P., eds, Proc. SPIE Conf. Ser. Vol. 7015, Adaptive Optics Systems. SPIE, Bellingham, p. 70153I

This paper has been typeset from a TEX/LATEX file prepared by the author.